We explored post training optimization techniques for improving the development performance of object detection architectures such as YOLOv3 on resource constrained devices such as the Jetson Nano. Through the TensorRT framework we applied optimization techniques such as Loop Fusion, Kernel Tuning, and Quantization and measured the impact on performance metrics like accuracy and inference execution time.

These techniques all function around trying to reduce the memory and computational overhead of the model. For example, quantization does so by reducing the precision of the variables used in inference allowing for both faster computations (given the correct HW) and reduced memory.

We used the TensorRT framework from NVIDIA to apply these optimization techniques to YOLOv3 models of different sizes, trained on different datasets and deployed on different types of hardware.

Future Work

- Repeat optimizations with a different framework to validate the accuracy drop measured by TensorRT.
- Utilize more aggressive INT8 quantization to examine more further examine trade-off of precision vs. performance.

References


Discussion

From the central figures, we see that our baseline measurements of YOLOv3 are as expected, roughly reproducing the mAP results reported by the original YOLOv3 paper. On both the RTX and Nano we pay significant speed costs, approximately 2x and 3x respectively, for marginal increases in mAP. The Nano fares worse given it’s limited compute power and resulting sensitivity to increases in FLOPS.

First we measured YOLOv3 performance with for non-quantization optimizations like layer fusion (LF) and kernel tuning (KT). The speed increased significantly (2-3x for RTX and 1.2x for Nano) but the mAP score dropped significantly. This was quite surprising, since the applied optimizations should only work to increase arithmetic intensity and reduce loads and stores to memory. Interestingly, a review of NVIDIA discussion boards indicated that this may be due to a bug in TensorRT’s implementation.

Next, we measure performance including 16-bit floating point quantization. We observed a negligibly small decrease in mAP with another significant boost to inference speed (~2.5x for RTX and Nano). This result is reasonable as we would expect that the error introduced by precision loss to be minimal in a large NN like YOLOv3 since less importance will be given to any one weight value. Furthermore, reducing precision loss will have a greater impact on small weight values (due to the nature of floating point), but these values inherently contribute less to the output calculation.

Overall, we found that applying optimization techniques such as quantization could provide for a 2x – 6x increase in inference execution speed. Such performance gains could be critical for real world deployment on an edge device. Of course, in our testing this came with a significant drop in accuracy, which may ultimately outweigh such speed gains.